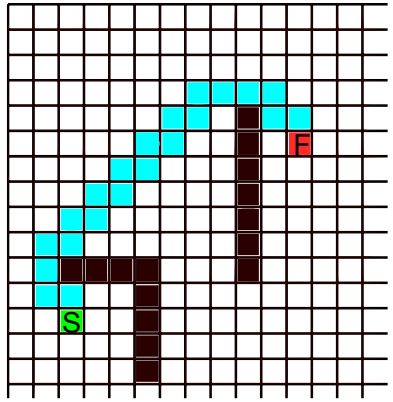
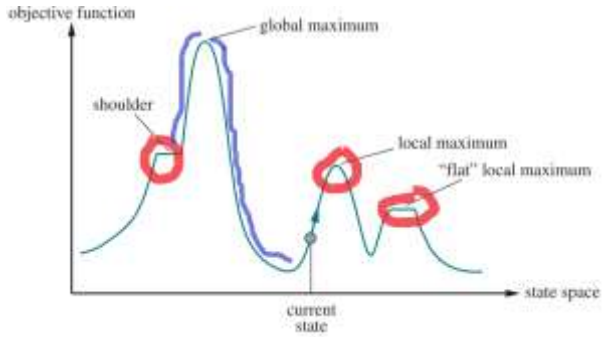
Searching Game Trees

ITI0210, lecture 5 (2021)

Review

Problem	Lecture	Look for	Method
Pathfinding	2-3	Steps to goal state	Brute force or heuristic search
Optimization	4	Refined state	Local search and metaheuristics



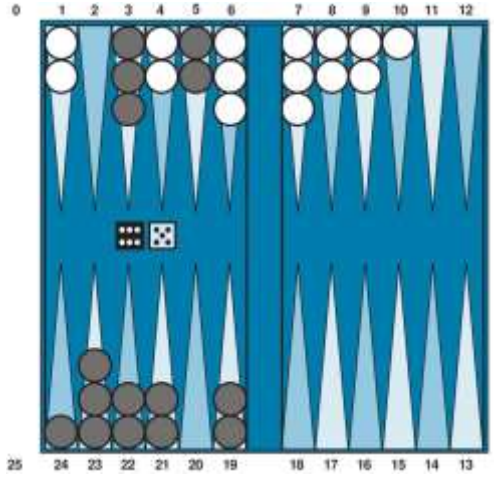


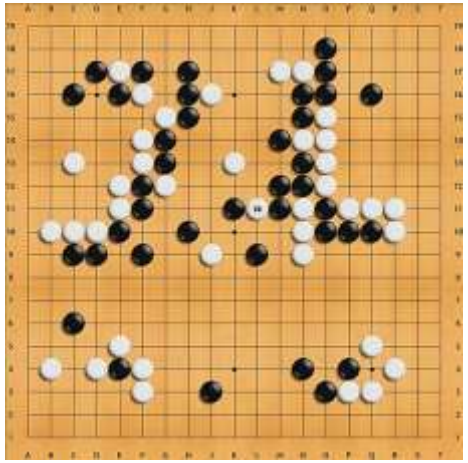
Acting Against Opponents

The opponent:

- Reacts to, and
- Undermines our actions

Traditional field of study: games





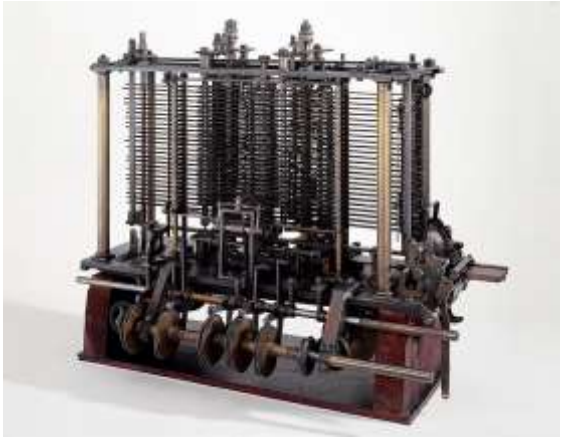
The Fascination With Games

Game-playing AI is an old idea

"**The Turk**" – a fake from 18th century

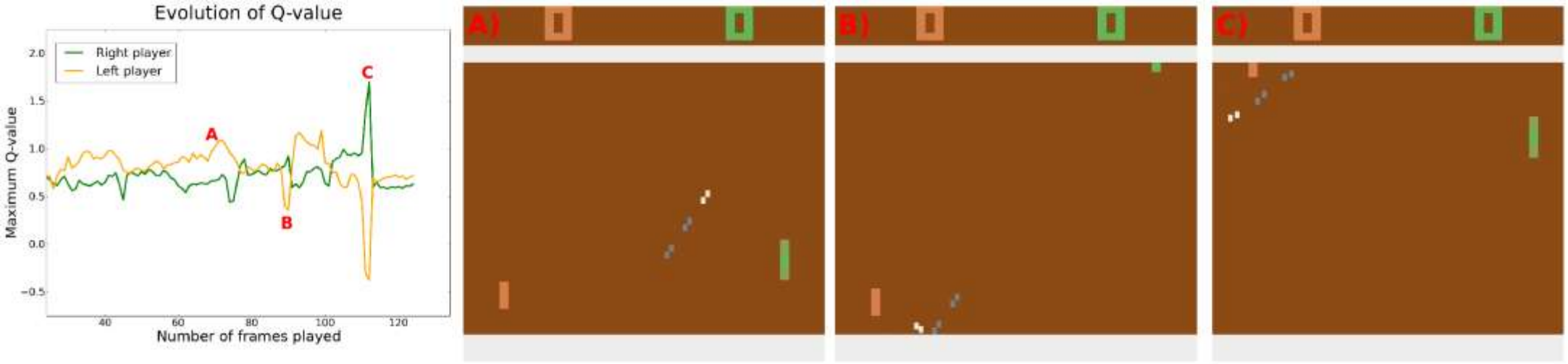
(there was a guy inside)

"*the combinations involved in the **Analytical Engine** enormously surpass any required, even by the game of chess.*" – Charles Babbage
(only partially built)



Benefits (Back to present day)

Example: learning to act in multi-agent environments
by training to play Pong



Tampuu A, Matiisen T, Kodelja D, Kuzovkin I, Korjus K, Aru J, et al. (2017) Multiagent cooperation and competition with deep reinforcement learning. PLoS ONE 12(4): e0172395

Goals

The obvious:

- Pick the best move, or action

But traditionally:

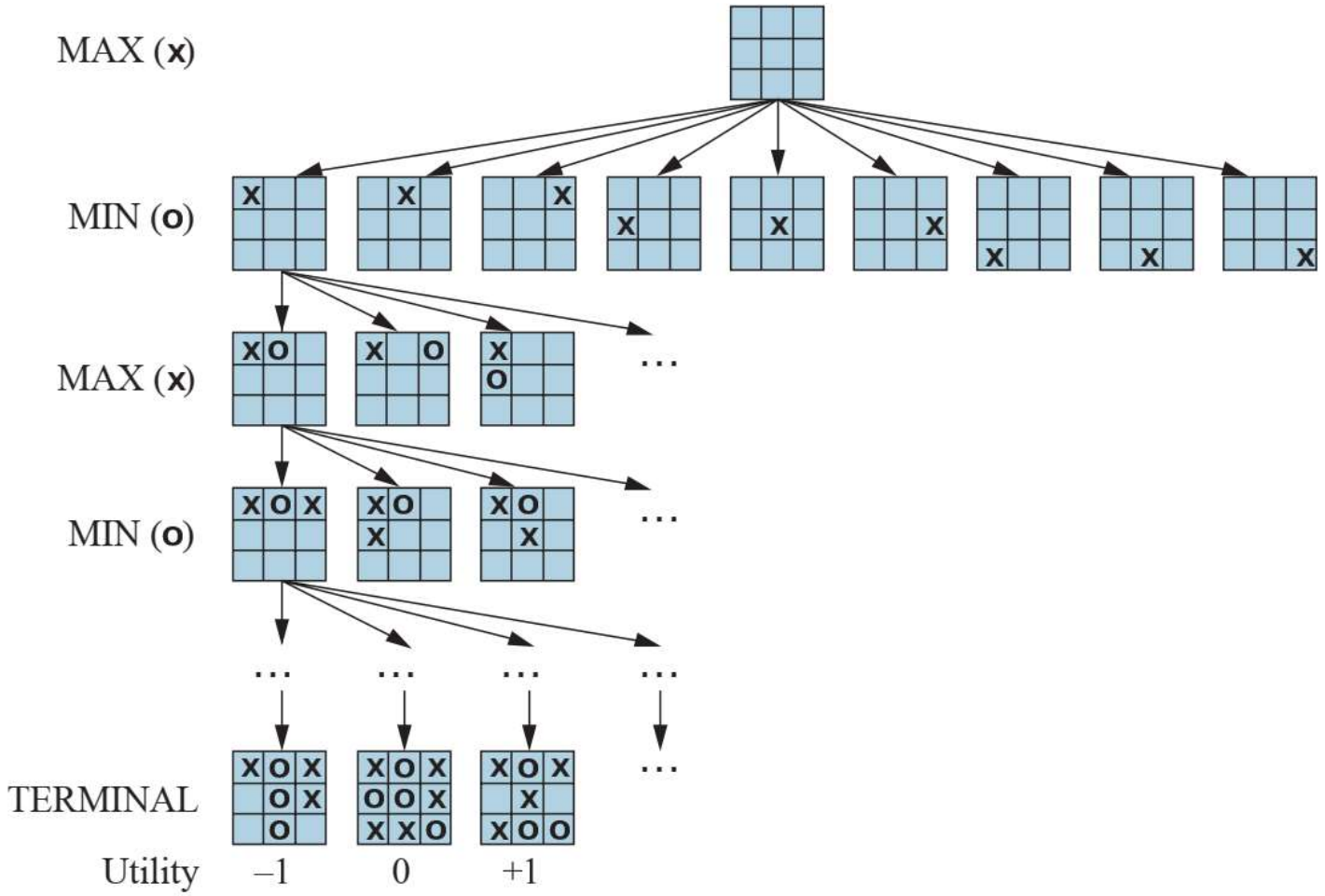
- Best move and the **proof that it is the best**

Minimax search

Proving the best move

Minimax

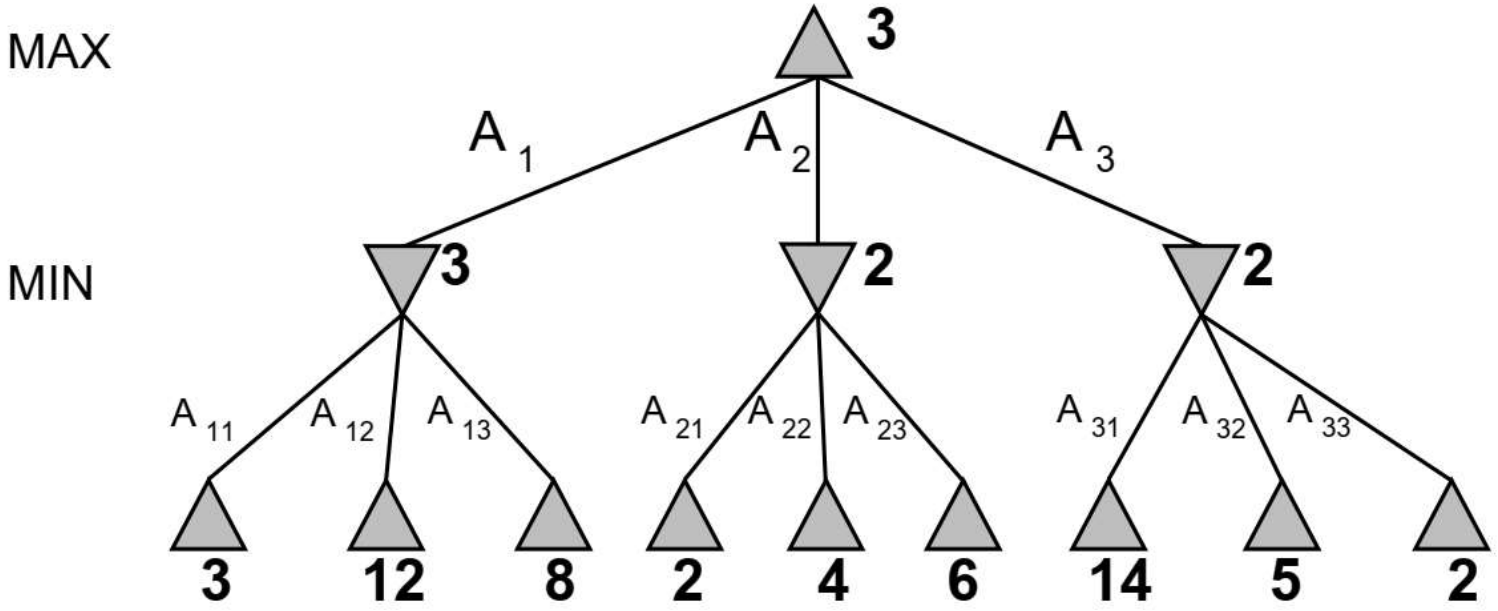
Solving tic-tac-toe



Minimax propagation

My turn: what is the best that I can do? (MAX)

Opp turn: what is the worst that can happen? (MIN)



Minimax

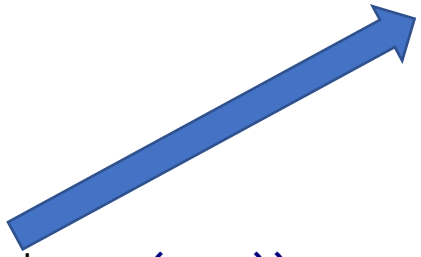
```
INF = 99999999


def minimax_search(pos):
    best_move = None
    best_val = -INF
    for move in pos.moves():
        val = min_value(pos.makemove(move))
        if val > best_val:
            best_move = move
            best_val = val
```

Minimax

```
INF = 99999999

def minimax_search(pos):
    best_move = None
    best_val = -INF
    for move in pos.moves():
        val = min_value(pos.makemove(move))
        if val > best_val:
            best_move = move
            best_val = val
```

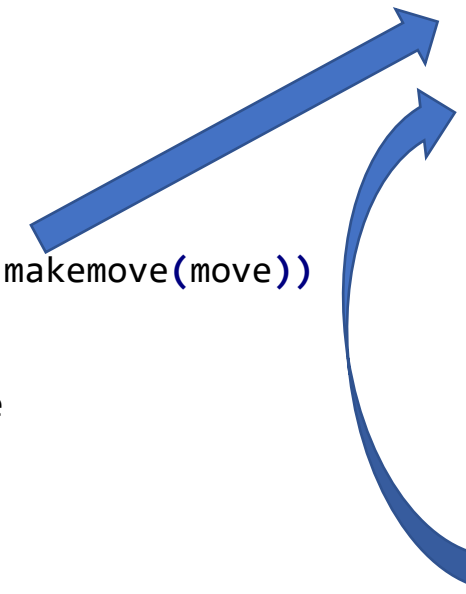


```
def min_value(pos):
    if pos.gameover():
        return pos.value()
    val = INF
    for move in pos.moves():
        val = min(val, max_value(pos.makemove(move)))
    return val
```

Minimax

```
INF = 99999999

def minimax_search(pos):
    best_move = None
    best_val = -INF
    for move in pos.moves():
        val = min_value(pos.makemove(move))
        if val > best_val:
            best_move = move
            best_val = val
```



```
def min_value(pos):
    if pos.gameover():
        return pos.value()
    val = INF
    for move in pos.moves():
        val = min(val, max_value(pos.makemove(move)))
    return val

def max_value(pos):
    if pos.gameover():
        return pos.value()
    val = -INF
    for move in pos.moves():
        val = max(val, min_value(pos.makemove(move)))
    return val
```

Minimax Tree Size

Exercise: how many tic-tac-toe games?

(Simple solution: unique move sequences $9!$,
smart solution: subtract mirrored positions, transpositions, early wins)

Chess: b=35, m=100, DFS tree $O(b^m)$ (atoms in universe only $\sim 10^{82}$)

Tic-tac-toe is an exception

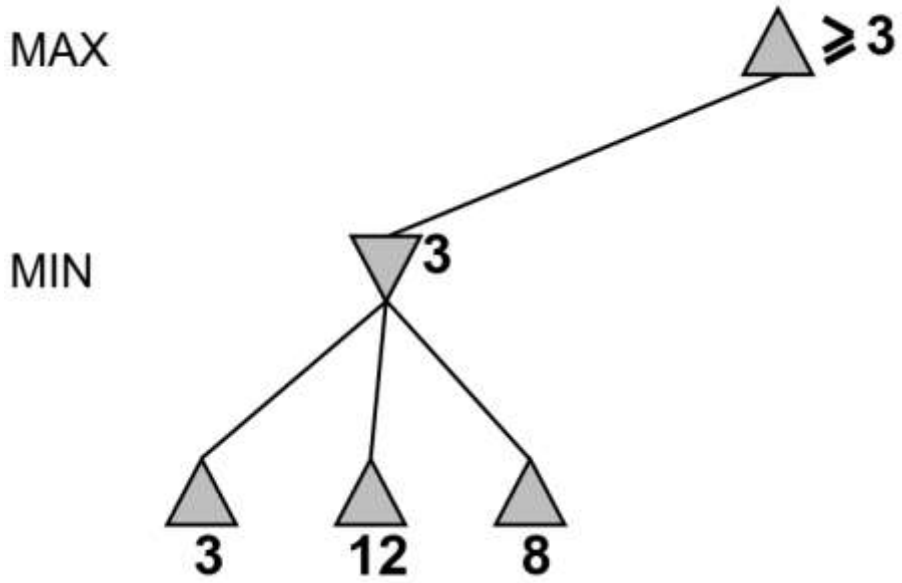
normally, not possible to search to the final positions

Dealing With Large Trees

- Some branches are not needed (*alpha-beta* algorithm)
- Limit search depth, guess the outcome (`eval()`)

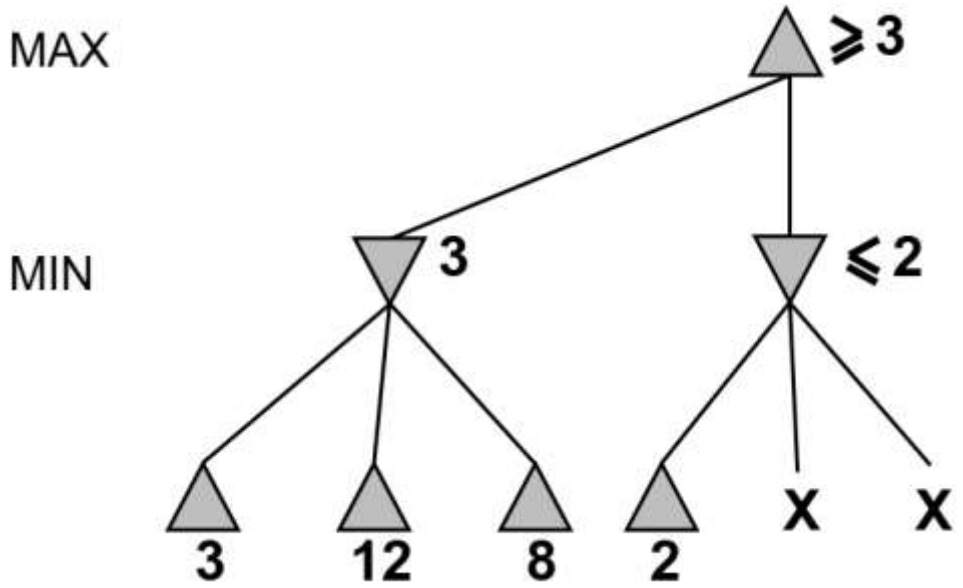
Alpha-Beta Search

Partially searched tree – score is **at least** 3



Alpha-Beta Search

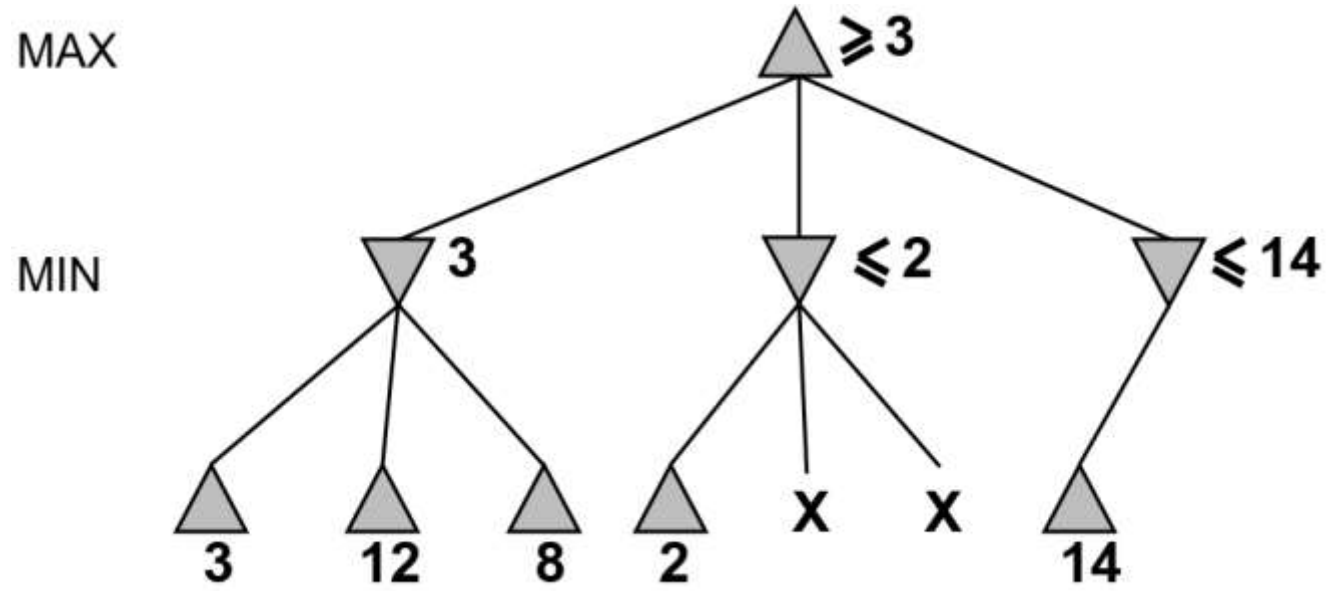
Current subtree will be **at most** 2



we will not make this move, stop searching it!

Alpha-Beta Search

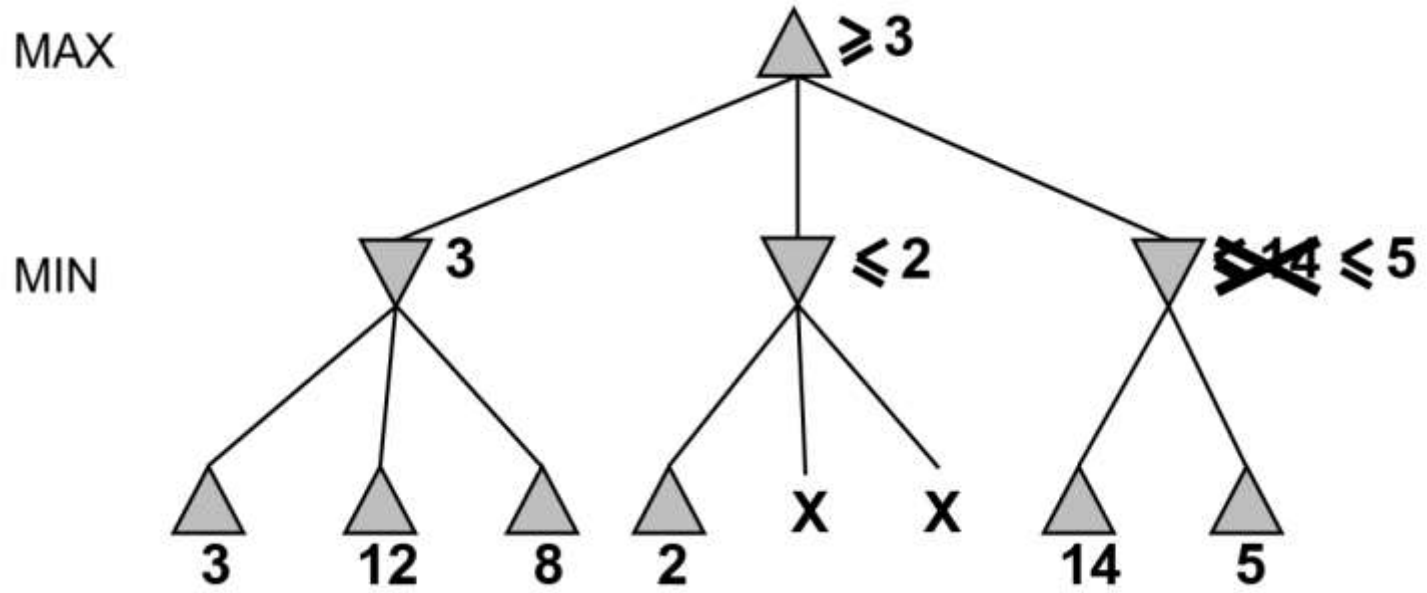
Pruning subtrees depends on search order



Best moves must be searched earlier

Alpha-Beta Search

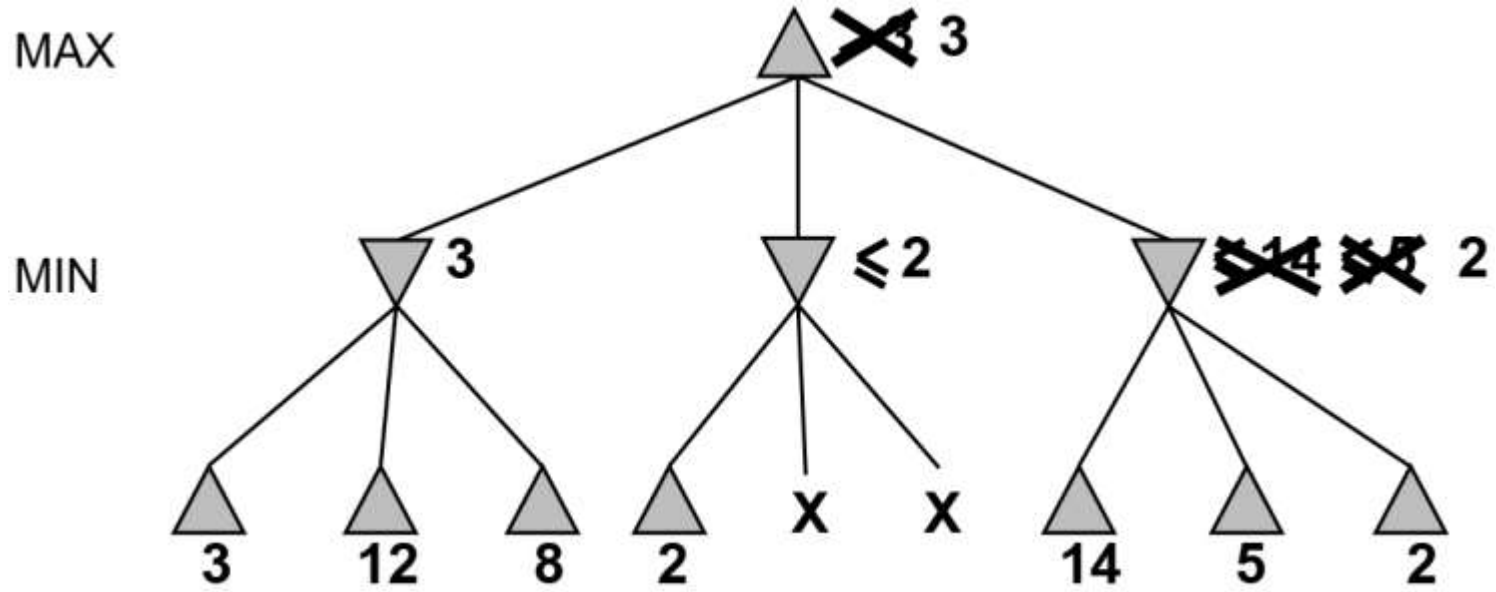
Pruning subtrees depends on search order



Best moves must be searched earlier

Alpha-Beta Search

Pruning subtrees depends on search order



Best moves must be searched earlier

Alpha-Beta Search

Doubles the search depth with perfect move ordering

e.g. can search to 12 half-moves where minimax searches to 6

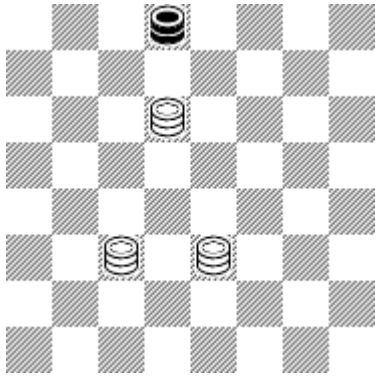
HUGE difference in tree size

With modest $b = 5$, $\frac{5^{12}}{5^6} = 15625$; **99.994%** of tree pruned by $\alpha\beta$

Change in actual playing strength depends on the game

Tricks of the Trade

- Eval: look at the position, guesstimate who is winning
- Save results of partial searches: transposition tables
- Make a database of best opening moves
- Make a database of all endgame positions



This was enough to “solve” checkers

Schaeffer, J.; Burch, N.; Y. Björnsson; Kishimoto, A.; Müller, M.; Lake, R.; Lu, P.; Sutphen, S. (2007). "Checkers is Solved". *Science*. **317** (5844): 1518–22

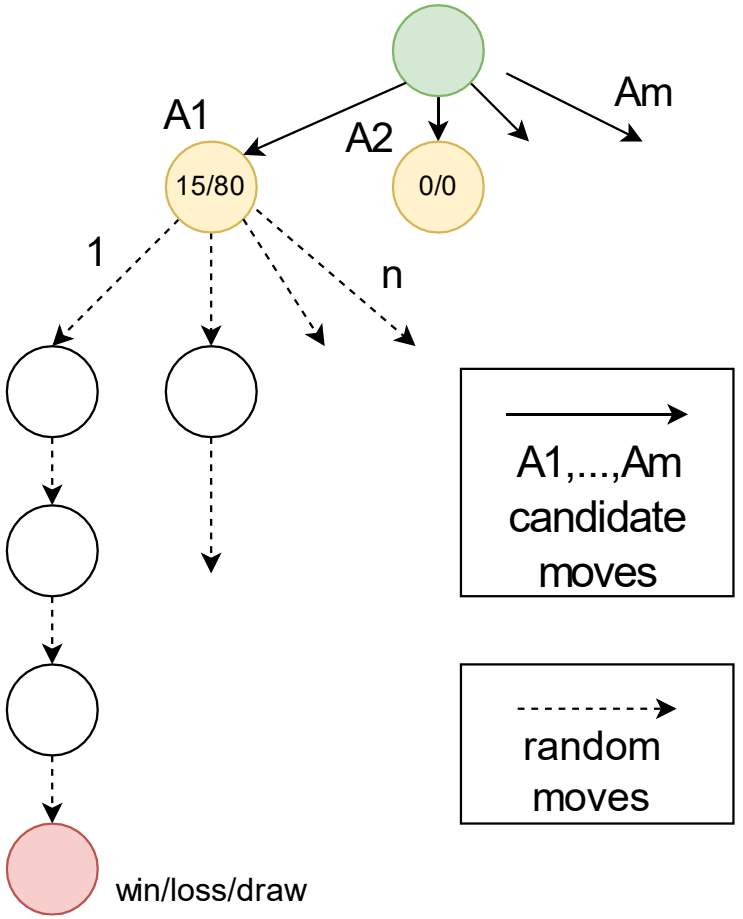
Monte Carlo Tree Search

The best move through probability

Sampling the Game Tree

A simple idea, good enough for your homework:

1. Play n completely random games from each move in current position
2. Pick the move with highest number of wins (or lowest # of losses)



Guiding the Search

Improvement: try to focus on best moves

- Spend less time on nonsense
- If we mistakenly think they are good, focusing on them will test that

Guiding the Search

Upper Confidence bound applied to Trees (UCT)

Select the tree node that has the highest score

$$UCB1(n) = \frac{U(n)}{N(n)} + C\sqrt{\frac{lnN(p)}{N(n)}}$$

n – node; p – parent

$U(n)$ – utility, e.g. number of wins (player to move matters!)

$N(n)$ – number of visits to n

Guiding the Search

Upper Confidence bound applied to Trees (UCT)

Select the tree node that has the highest score

$$UCB1(n) = \frac{U(n)}{N(n)} + C\sqrt{\frac{lnN(p)}{N(n)}}$$

Exploitation term

Tweaking parameter

Exploration term

UCB1

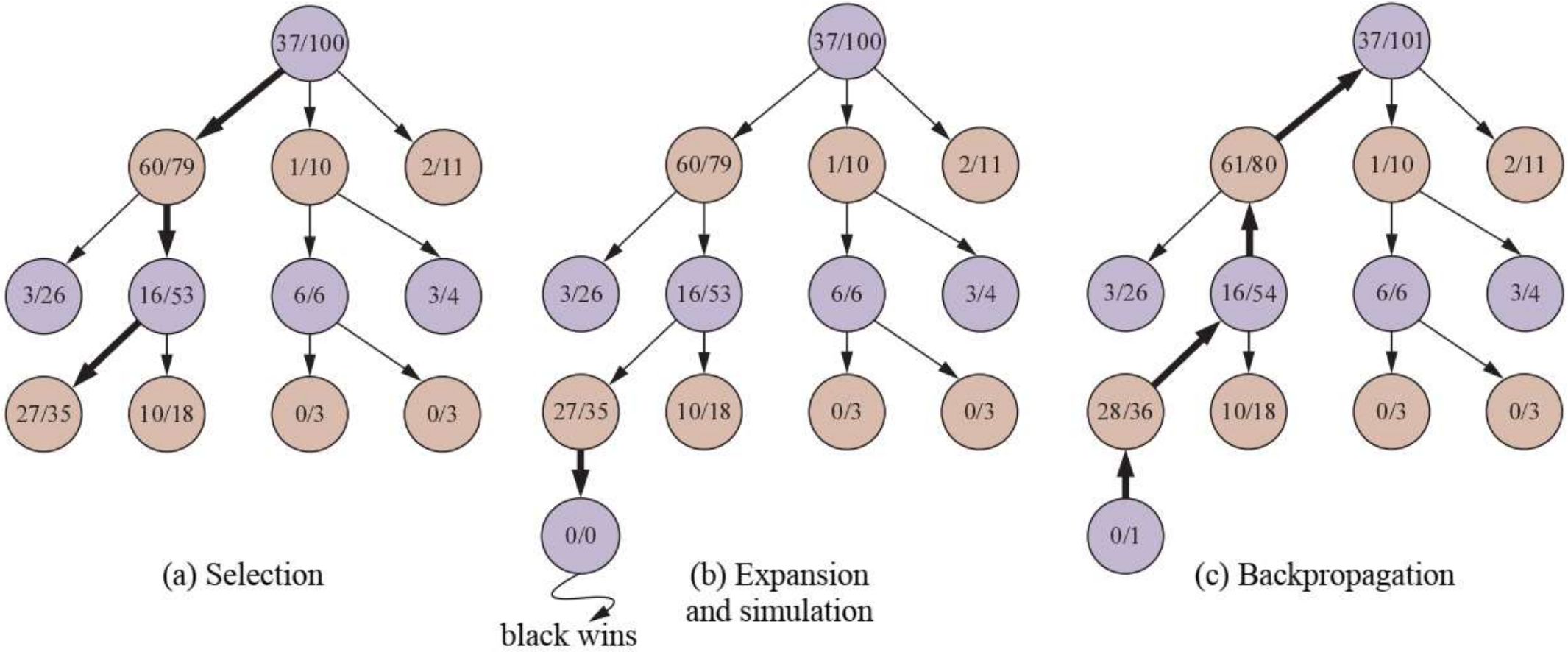
Exploitation: prefer nodes with high winning % $\frac{U(n)}{N(n)}$

Exploration: prefer least visited children $\sqrt{\frac{lnN(p)}{N(n)}}$

Tweak the best balance with C (suggested by theory: $C = \sqrt{2}$)

Can add more stuff, like move probability from a deep NN

MCTS with UCT



MCTS with UCT

Tree consists of:

- Growing permanent part with score/visits statistics
- The random simulations or “rollouts” – throwaway

Search tree expansion explained:
https://www.youtube.com/watch?v=UXW2yZndl7U

Video notes:

- score is “generic” – in games we care more about wins/losses
- other ways of generating children are possible

MCTS with UCT

Deciding the move from root position:

one leading to child node with highest $N(n)$

Because of the UCB1 formula, it will also have high $\frac{U(n)}{N(n)}$

Exercise: Why?

General idea: 65/100 is a safer choice than 2/3